Bots and Boxes: AlphaZero for Playing Dots and Boxes

Christos Polzak
Department of Computer Science
Stanford University
clcp@stanford.edu

Renee Qin
Department of Computer Science
Stanford University
reneeqin@stanford.edu

Frank Zhao
Department of Computer Science
Stanford University
frankz24@stanford.edu

Abstract

The game of Dots and Boxes, a classic children's pastime played worldwide, conceals surprising strategic complexities beneath its simple rules. Its exponentially growing state space necessitates sophisticated strategic reasoning and long-term planning. In this project, we implement an AlphaZero-based approach from the ground up, training models capable of excelling across three board sizes. Our trained agents outperform random and greedy baselines as well as human players. We analyze the effects of data augmentation and network architecture, as well as trade-offs among key hyperparameters on model learning. Code is on GitHub.

1 Introduction

We aim to design an intelligent autonomous (i.e., computer) opponent for the classic pen-and-paper game of Dots and Boxes. The game begins with a blank grid of dots and the players take turns drawing horizontal and vertical lines between adjacent dots. If a player completes all four sides of a 1x1 square (box) by drawing the final line, they "capture" the box, earning one point, and get an additional turn. The player who completes the most boxes by the end of the game is declared the winner. In the example shown in Figure 1, the blue player is the final winner.

Beneath these simple rules, strategic complexity arises from decisions about when to complete boxes versus when to set up opportunities while avoiding moves that benefit the opponent. In fact, given a board with $n \times n$ boxes, there are $2n(n+1)$ edges, which lead to $2^{2n(n+1)}$ possible states. This exponentially growing state space makes the game extremely challenging and interesting. Notably, the largest solved configuration of Dots and Boxes that we are aware of is the 4x5 grid, as presented by Barker and Korf (2012).

To give an example of the game's complexity, consider the intermediate board state in Figure 1. It is the blue player's turn, and it appears to be very tempting to complete the middle box and then complete the top middle box. However, after completing these two boxes, the blue player must draw another edge, which would lead to the red player completing the rest four boxes and win the game. Alternatively, the blue player can draw the top middle edge without completing any boxes and force a win. This illustrates that long-horizon planning is essential in this game.

Our goal is to design a system capable of playing a highly effective strategy for the Dots and Boxes game of various sizes. We believe this is interesting because determining the optimal policy demands substantial long-term planning and strategic reasoning. The rewards associated with lines drawn at the beginning of the game only become apparent many turns later at the end of the game. Given the

Style from the 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

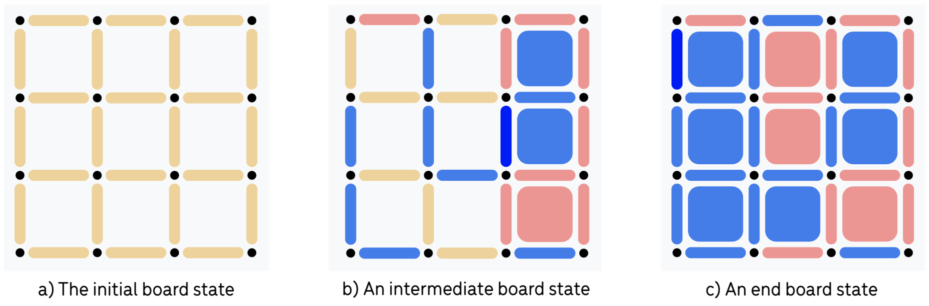

Figure 1: **The Dots and Boxes Game** refers to the game involving a grid of dots, where players take turns to mark edges in the goal of completing as many boxes as possible.

complexity and nuances of the game and its large state space, we believe the best approach is learning through self-play, akin to the seminal work by Silver et al. (2017). Due to computational constraints, we focus on games with 2×2 boxes, 3×3 boxes, and 4×4 boxes. The 3×3 game is an interesting case because we know it has been theoretically solved (Numberphile, 2015) and the second player can always win.

Overall, our project implements an AlphaZero (Silver et al., 2017) approach from scratch and demonstrates its success across three board sizes. We further analyze this approach by running ablation studies and tweaking key hyper-parameters.

1.1 Problem Formulation

This game can be modeled as a fully observable Markov decision-making process. Our environment consists of a grid of dots that can be connected by line segments between each of the dots. The grid can be different sizes of n by n boxes, where $n >= 2 \in \mathbb{Z}$. Our problem is sequential, as each decision to place a line is dependent on the game's current state and will affect future decisions. The decisions are not independent, and each line drawn changes the potential actions available in subsequent turns. As such, the game progresses through a series of turns until no further lines can be drawn.

The state of the MDP is defined as the configuration of lines on the grid and the boxes completed. The action space at each state consists of all valid lines that can be drawn. Transitions in this game are deterministic: drawing a line directly modifies the grid state without randomness. The reward structure could involve points being earned by capturing boxes during the game, but the ultimate reward is based on the final score difference, as the objective is to win the game by having more boxes than the opponent.

1.2 Sources of Uncertainty

The primary source of uncertainty in Dots and Boxes is the opponent's actions. While the game state is fully observable, predicting the opponent's intentions or strategy introduces significant complexity. The optimal decision-making process requires the player to anticipate and counteract the opponent's moves, which may involve analyzing potential responses across multiple future turns. An effective agent must, therefore, incorporate the ability to model and adapt to the opponent's strategies dynamically.

Our work aims to train an algorithm that, given the current state, produces an optimal policy to maximize the likelihood of winning. The algorithm should account for the delayed rewards and strategic interactions inherent to the game.

2 Literature Review

The game of Dots and Boxes has been extensively studied within the domain of combinatorial game theory, primarily focusing on endgame analysis and optimal strategies. Early foundational work by Berlekamp (2017) formalized concepts such as double-crosses and chain counting, which are essential for understanding advanced strategies in the game. These insights provided a theoretical framework for analyzing the dynamics of Dots and Boxes, particularly in identifying optimal moves during critical stages of the game (Berlekamp, 2020; Guy and Berlekamp, 1991).

More recently, computational approaches have focused on solving specific board configurations. Barker and Korf (2012) leveraged techniques to determine the optimal strategy for a 4x5 grid. Their work demonstrated the feasibility of solving larger grids but highlighted the exponential growth of computational complexity as grid sizes increase. These findings underscore the challenges associated with generalizing solutions to arbitrary board sizes, making the development of scalable algorithms a significant area of interest.

Reinforcement learning has emerged as a promising methodology for tackling games with large state and action spaces. The AlphaZero algorithm, introduced by Silver et al. (2018), demonstrated remarkable success in mastering games such as Go, Chess, and Shogi through self-play and deep reinforcement learning. There have also been other studies on this algorithm (Bratko, 2018; McGrath et al., 2022; Tian et al., 2019; Zhang and Yu, 2020). However, the application of AlphaZero to games like Dots and Boxes remains underexplored.

Prior work in related domains often relies on handcrafted evaluation functions or game-specific heuristics to guide decision-making. For example, strategies in games such as Hex and Connect Four have incorporated domain knowledge to improve computational efficiency (Brown and Sandholm, 2019). In contrast, the generalizability of AlphaZero allows for a broader exploration of strategies without predefined heuristics, making it a suitable candidate for a game like Dots and Boxes, where optimal play involves complex trade-offs between immediate and long-term rewards.

This work builds upon the literature mentioned above by adapting AlphaZero to various board-size versions of Dots and Boxes. Our focus is on addressing the unique challenges posed by the game, including delayed rewards, opponent modeling, and the combinatorial explosion of state-action spaces. By leveraging self-play and deep reinforcement learning, we aim to extend the applicability of AlphaZero to strategic games characterized by intricate decision-making processes.

3 Methodology

3.1 AlphaZero Algorithm

AlphaZero is a general algorithm for game-playing that hybridizes actor-critic methods and Monte Carlo Tree Search (MCTS). The main component is a trainable neural network $f(s) = (\pi, v)$ that predicts a policy $\pi(s)$ as a probability distribution over the possible moves, as well as a value $v(s)$ representing how favorable the current board is (+1 means certain victory, -1 means certain defeat). In each learning iteration, AlphaZero first plays games against itself: it uses $\pi(s)$ (in conjunction with a variation of UCB1 and Dirichlet noise to encourage exploration) to perform rollouts from the current move and uses $v(s)$ to build up an estimate for $Q(s, a)$ at each board state, then selects a final action for the current turn by using the number of rollouts for each action (i.e. $N(s, a)$) as a probability distribution over which to select a move; it then repeats this process on the next move (playing as the other player) until the game ends. After multiple self-play games, the $N(s, a)$ counts are treated as the "true" probability distribution $\tilde{\pi}$ for each turn, and the final results of the games are used to assign the "true" value $\tilde{v}$ of each board (-1 if that player lost, +1 if the player won, and 0 if there was a draw). After these self-play games, the neural network is then fitted to learn with a dataset of $(s_i, \tilde{p}i_i, \tilde{v}_i)$ tuples taken from the self-play games. Finally, the neural network is pitted against the previous best version of the network, and the learned changes are discarded if the new network does not outperform the old version. This process repeats back and forth until the model converges (for example, until new versions cannot beat old versions of the network, or in our case, until performance against a known baseline stabilizes).

3.2 Game State Representation and Model Architecture

We choose AlphaZero because of its prior success in games with intractably large action spaces and delayed, complex rewards; more concretely, it seems like a good fit because despite its large state and action spaces, Dots and Boxes is a highly structured game with large amounts of symmetry and sub-structure, and with AlphaZero, we are able to model the game in line with our inductive biases, as well as use a neural network to learn complex but structurally relevant strategies. Concretely, we observe that a large Dots and Boxes board contains numerous sub-boards that can contain useful structure (for example, a nearly-complete 2x1 sub-board presents a player with the option to essentially swap who can fill which future chains of boxes). We exploit this by representing our game board as 4-channel binary image, where each pixel represents a box in the grid, and each channel represents one of the sides of the box (e.g. `[1, 0, 0, 1]` means that the top and left edges of the box are filled). In addition to the board itself, we also give to the model the current percentage of the total number of boxes each player has filled. Note that without this information, all final boards would be identical in state; additionally, note that the current state needs to depend only on how many boxes each player has, not which exact ones are assigned to them.

To learn from the substructure present in the "image," we leverage convolutional neural networks, which have kernels that can be used as 2D sliding windows over an image and thus directly let us build up an interperetation of the current board based on substructure. Specifically, we use a highly scaled-down version of the popular ResNet architecture He et al. (2015) with only 4 convolutional blocks, modify its output to take in the score state, and then use that output to predict both the policy π and value v. Since neural networks and image processing are not the focus of this class, we mention this only briefly and encourage the interested reader to explore the literature on their own.

4 Results

In this section, we introduce the baselines in Section 4.1, present the main quantitative results in Section 4.2, and conduct some analysis in Section 4.3.

4.1 Baselines

We introduce the baselines that are compared against our model below.

Random Baseline. The random baseline always randomly samples a valid action to play. It represents chance and is the weakest baseline in this project.

Greedy Baseline. The greedy baseline always seeks to complete a box whenever possible. Otherwise, it randomly samples a valid action that does not immediately set up its opponent to score (if possible). This baseline serves as a good proxy for a novice human player. As we have seen in Section 1, it is not optimal.

General AlphaZero Implementation. The popular Github repository (Thakoor et al., 2016) contains an AlphaZero implementation for the Dots and Boxes game. However, since they only release a model for the 3×3 board size, we only compare our model with it on a 3×3 board.

Human. Each of us play against the models we trained.

4.2 Quantitative Results

In Figure 3, we show the training progress of our models on the three board sizes. As we can see, the training losses steadily drop and show signs of convergence in all three cases. The winning rate against the random and greedy baselines consistently improves until stabilizing around a high number. It is worth noting that our model does not achieve perfect win rates because the game can tie in the 2×2 and 4×4 cases and it is not always possible to force a win depending on which model goes first.

Next, we compare our trained model against the baselines described in Section 4.1. For humans, each of us played 6 games with the model while all other numbers are produced by playing 100 games with alternating starting players. As we can see in Table 1, our model consistently beats the random and greedy baselines. It also ties with the AlphaZero implementation by Thakoor et al. (2016) on the 3×3 board. Since the optimal second player can force a win on the 3×3 board, it may indicate that both models have learned the optimal strategy. As the board size increases, we, as human players, find it increasingly difficult to beat the model.

	Win Rate Against			
Board Size	Random	Greedy	General Alphazero	Human Player
2 x 2	0.910	0.710	N / A	4+4.5 / 6+6
3 x 3	1.000	0.820	0.500	5+5 / 6+6
4 x 4	1.000	0.770	N / A	6+6 / 6+6

Table 1: **Quantitative Results.** We compare our trained model against baselines outlined in Section 4.1. Our model consistently beats random and greedy baselines as well as human players, while tying with the reference implementation.

4.3 Analysis

In this section, we further analyze our approach through ablation studies and tweaking key hyperparameters. Figure 2 shows the winning rate against the greedy baseline as training progresses for different model variations. Table 2 shows the quantitative numbers when playing these variations against our model.

Augmentation. In the dots and boxes game, the board has 8 unique symmetries through rotation and reflection. These symmetries represent equivalent states. In our implementation, we augment the self-play data by producing these symmetries. We find that this approach significantly helps model learning without introducing significant overhead. Indeed, without the augmentation, the model improves much slower with the same number of iterations and performs much worse compared to our full model with augmentation.

Network Backbone. We chose to use a CNN-based architecture because we believe the spatial information of the edges is important to understand the game and play it well. We ablate this backbone by replacing it with a fully connected network (referred to as “MLP Network"). We replace each CNN layer and residual block with a linear feed-forward layer, resulting in 6 linear layers with 256 hidden dimensions each. In addition, we add Dropout (Hinton et al., 2012) to these linear layers with $p = 0.3$ following Thakoor et al. (2016). The total parameter count for the CNN-based backbone is 390k, whereas the total parameter count for the fully connected backbone is 350k. Thus, the comparison should be reasonably fair. We see that the fully connected backbone achieves similar performance as the CNN-based model against the greedy baseline. However, the CNN-based model still outperforms the fully connected one by a small margin in direct head-to-head. We believe our backbone would be a clearer favorite in larger game boards that are harder to memorize.

Thin Deep Trees. For this variation, we reduce the number of self-play games from 96 to 32 while increasing the number of MCTS simulations from 100 to 500. We want to study whether we can trade self-playing with more simulation. For a fair comparison, we let the model run 300 iterations instead of 100 so that all models see roughly the same amount of data overall. It appears that having more self-play data in each iteration is still more important.

Exploratory Deep Trees. Similar to “thin deep trees", the number of self-play games is reduced while the number of MCTS simulations is increased. However, we also increase exploration by increasing the `cpuct` and `noise` hyperparameters in MCTS. This helped the model to achieve the same performance as our reference model. As such, if we have less self-play games in each iteration, increasing exploration might be important.

	Win Rate Against			
Board Size	Without Augmentation	MLP Network	Thin Deep Trees	Exploratory Deep Trees
3 x 3	0.880	0.610	0.700	0.500

Table 2: **Comparison with Other Model Variations.** We find that 1) symmetry augmentation drastically helps our model, 2) a CNN-based backbone has a slight edge over purely fully-connected layers, and 3) trading self-play games with more MCTS simulations worsens performance but can potentially be compensated by more exploration.

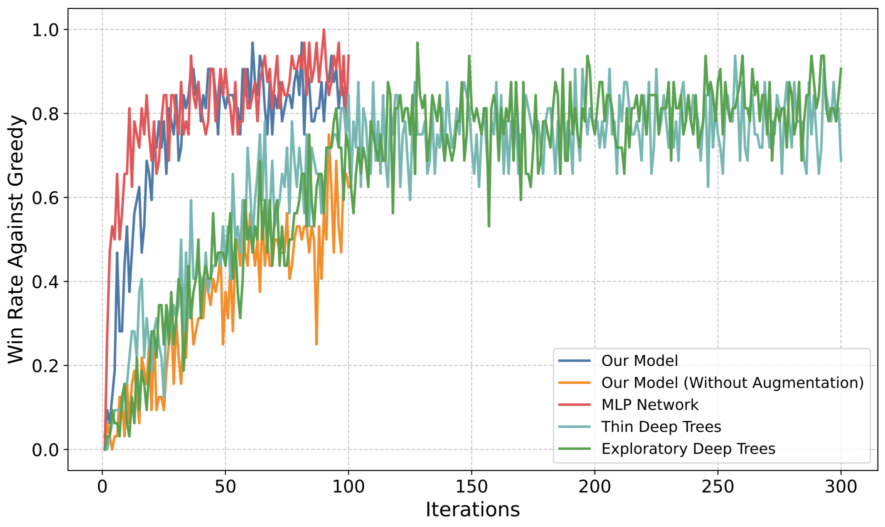

Figure 2: **Comparison Against Other Model Variations.** We analyze the training progress in terms of winning rate against the greedy baseline for different model variations. We find that data augmentation significantly helps model learning, while the different architectures show little difference. In addition, when we increase the number of MCTS simulations and decrease the number of self-play games per iteration, the model requires more iterations to converge.

5 Discussion

5.1 Limitations

While our approach demonstrates the potential of AlphaZero in becoming adequate at playing Dots and Boxes, several limitations exist. Firstly, the computational resources required for training are significant, especially as the grid size increases beyond 4x4. This constraint limits the scalability of our method to larger board configurations, which may require additional optimization or approximation techniques. Secondly, our model's performance heavily depends on hyperparameter tuning and the design of neural network architectures. Small changes in these parameters can impact the learning process, introducing variability in outcomes. Moreover, the interpretability of the learned strategies remains a challenge, as the neural network's decisions are not easily decomposable into human-understandable heuristics. As a result, it is difficult to determine whether the model has truly learned the optimal strategy, when one exists.

5.2 Future Work

We believe bringing the algorithm to much larger boards would be both interesting and challenging. On one hand, no one has determined whether an optimal strategy exists for the bigger boards and it would be exciting to make progress towards it. On the other hand, computational constraints become a problem, and it is more difficult to properly evaluate model performance for bigger boards. One direction to alleviate the computational constraints may be to fine-tune trained CNN-based state embedders on small boards for bigger boards, as commonalities certainly exist among boards of different sizes. Another area of future research is improving interpretability by integrating explainable AI techniques. This could involve visualizing the agent's decision-making process or identifying patterns in its strategic preferences. Enhanced interpretability would not only help in debugging and refinement but also make the system more accessible to educational and recreational applications. We can also extend our methodology to other games with similar characteristics, such as delayed rewards and adversarial interactions, which could provide valuable insights into the generalizability of AlphaZero-based approaches.

NB: Please see Appendix A (after references) for group member contributions.

References

Joseph Barker and Richard Korf. Solving dots-and-boxes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):414–419, 2012.

Elwyn R. Berlekamp. *The dots and boxes game: Sophisticated child's play*. CRC Press, 2017.

Elywn R. Berlekamp. *Winning ways for your mathematical plays, volume 3*. CRC Press, 2020.

Ivan Bratko. Alphazero – what's missing? *Informatica (Slovenia)*, 42:7–11, 2018.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Richard K. Guy and Elwyn R. Berlekamp. *Combinatorial games*. American Mathematical Society, 1991.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Martin Wattenberg, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of chess knowledge in alphazero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, 2022.

Numberphile. How to always win at dots and boxes - numberphile, 2015.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Shantanu Thakoor, Surag Nair, and Megha Jhunjhunwala. Learning to play othello without human knowledge, 2016.

Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. ELF OpenGo: an analysis and open reimplementation of AlphaZero. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6244–6253. PMLR, 2019.

Hongming Zhang and Tianyang Yu. *AlphaZero*, pages 391–415. Springer Singapore, Singapore, 2020.

A Appendix

A.1 Contributions

We split the work evenly. Christos developed the game logic, state representation, and Monte Carlo Tree Search. Renee designed the model architecture and implemented data storage. Frank handled the main training and evaluation pipeline. All members contributed to hyperparameter tuning and refinements to our approach. For writing, Renee contributed the introduction, literature review, and discussion sections; Christos wrote the methodology; and Frank authored the results.

A.2 Additional Graphs

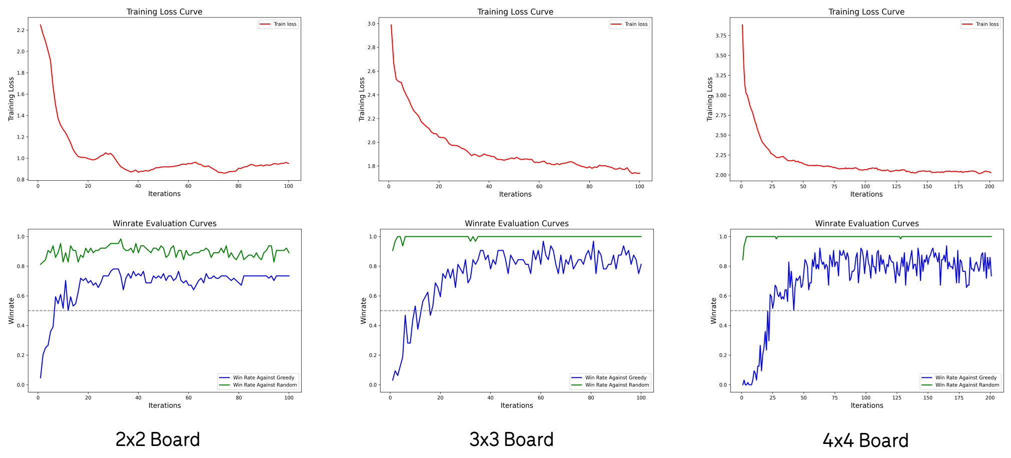

Figure 3: **The Training Progress for Different Size Boards.** We show the training progress of our approach on three different board sizes: 2 by 2 in the left column, 3 by 3 in the middle column, and 4 by 4 in the right column. The red curves in the top row show the training loss. The green curves in the bottom row show our model's win rate against a random baseline. And the blue curves in the bottom row show our model's win rate against a greedy baseline. As training progresses, our model quickly learns to beat both the random and greedy baselines.